PL Assignment #3 : Cute17 Scanner

과제물 부과일 : 2017-03-23 (목)

Program Upload 마감일 : 2017-03-29(수) 23:59:59

문제

token을 인식하여 token과 lexeme을 모두 출력하는 program을 작성하시오.

예를 들어, token으로 분류할 문자열이 아래와 같을 경우,

```
( define length
  ( lambda ( x )
    ( cond ( ( null? X ) 0 )
          ( #T ( + 1 ( length ( cdr x ) ) ) ) ) ) )
```

(주의: token과 token 사이에 반드시 공백이 있다.)

출력은 아래와 같아야 한다.

```
L_PAREN         (
DEFINE          define
ID              length
L_PAREN         (
LAMBDA          lambda
L_PAREN         (
ID              x
R_PAREN         )
L_PAREN         (
COND            cond
L_PAREN         (
L_PAREN         (
NULL_Q          null?
ID              x
R_PAREN         )
INT             0
R_PAREN         )
INT             0
R_PAREN         )
L_PAREN         (
TRUE            #T
L_PAREN         (
PLUS            +
INT             1
L_PAREN         (
ID              length
L_PAREN         (
CDR             cdr
R_PAREN         )
R_PAREN         )
R_PAREN         )
R_PAREN         )
R_PAREN         )
R_PAREN         )
R_PAREN         )
```

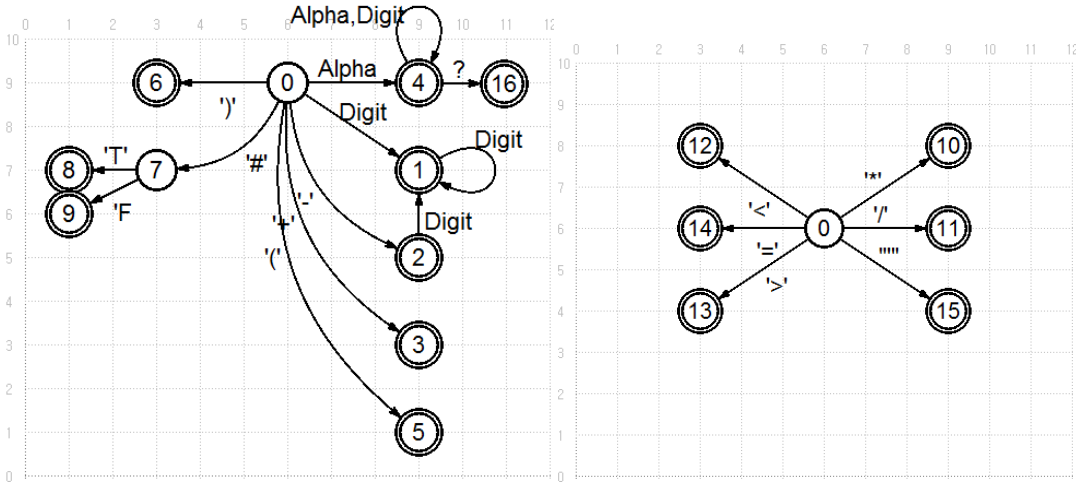
Regular Expression

```
QUESTION: ID '?'
ID:       Alpha[Alpha|Digit]*
INT:      Digit+ | PLUS Digit+ | MINUS Digit+

L_PAREN:  '('
R_PAREN:  ')'
PLUS:     '+'
MINUS:    '-'
TIMES:    '*'
DIV:      '/'
LT:       '<'
EQ:       '='
GT:       '>'
APOSTROPHE: '\''
TRUE:     Sharp 'T'
FALSE:    Sharp 'F'

Sharp:    '#'
Alpha:    [A-Z] | [a-z]
Digit:    [0-9]
```

- ID나 QUESTION중에서 특별한 의미를 가지는 keyword와 해당 token 이름은 다음과 같다.(keyword가 아니면 ID로 간주)

"define"	DEFINE
"lambda"	LAMBDA
"cond"	COND
"quote"	QUOTE
"not"	NOT
"cdr"	CAR
"car"	CDR
"cons"	CONS
"eq?"	EQ_Q
"null?"	NULL_Q
"atom?"	ATOM_Q

mDFA



작성해야 할 코드

1) transition matrix 작성

2) 토큰의 type 과 lexeme 초기화

Program 작성

Programming

```
from string import letters, digits, whitespace

class CuteType:
    INT=1
    ID=4
    MINUS=2
    PLUS=3
    L_PAREN=5
    R_PAREN=6
    TRUE=8
    FALSE=9
    TIMES=10
    DIV=11
    LT=12
    GT=13
    EQ=14
    APOSTROPHE=15
    DEFINE=20
    LAMBDA=21
    COND=22
    QUOTE=23
    NOT=24
    CAR=25
    CDR=26
    CONS=27
    ATOM_Q=28
    NULL_Q=29
    EQ_Q=30
    KEYWORD_LIST=('define', 'lambda', 'cond','quote', 'not', 'car', 'cdr',' cons', 'atom?',
'null?', 'eq?' )
```

```
#입력받은 문자열이 keyword인지 확인하는 함수
def check_keyword(token):
    """
    :type token:str
    :param token:
    :return:
    """
    if token.lower() in CuteType.KEYWORD_LIST:
        return True
    return False

#Keyword가 되는 문자열을 입력받아 type number를 반환
def _get_keyword_type(token):
    return {
        'define': CuteType.DEFINE,
        'lambda': CuteType.LAMBDA,
        'cond': CuteType.COND,
        'quote': CuteType.QUOTE,
        'not': CuteType.NOT,
        'car': CuteType.CAR,
        'cdr': CuteType.CDR,
        'cons': CuteType.CONS,
        'atom?': CuteType.ATOM_Q,
        'null?': CuteType.NULL_Q,
        'eq?': CuteType.EQ_Q
    }[token]

CUTETYPE_NAMES = dict((eval(attr, globals(), CuteType.__dict__), attr) for attr in dir(
    CuteType()) if not callable(attr) and not attr.startswith("__"))


class Token(object):
    def __init__(self, type, lexeme):
        """
        :type type:CuteType
        :type lexeme: str
        :param type:
        :param lexeme:
        :return:
        """
        #Fill Out
        # Initialize the token to generate.

        print "[" + CUTETYPE_NAMES[self.type] + ": " + self.lexeme + "]"

    def __str__(self):
        # return self.lexeme
        return "[" + CUTETYPE_NAMES[self.type] + ": " + self.lexeme + "]"

    def __repr__(self):
        return str(self)


class Scanner:

    def __init__(self, source_string=None):
        """
        :type self.__source_string: str
        :param source_string:
        """
        self.__source_string = source_string
        self.__pos = 0
        self.__length = len(source_string)
        self.__token_list = []

    def __make_token(self, transition_matrix, build_token_func=None):
        old_state = 0
```

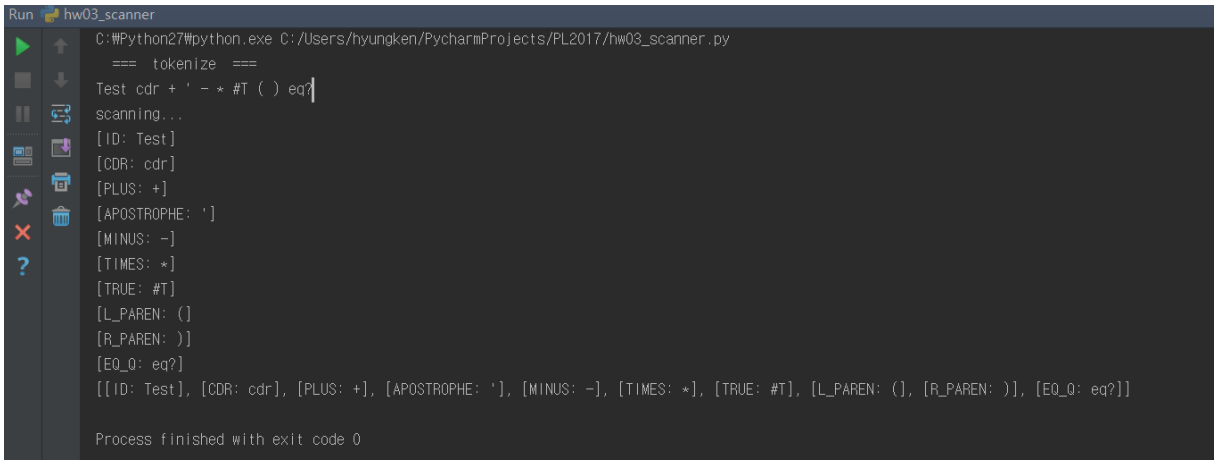
```
        self.__skip_whitespace()
        temp_char = ""
        return_token = ""
        while not self.eos():
            temp_char = self.get()
            if old_state == 0 and temp_char in (")", "("):
                return_token = temp_char
                old_state = transition_matrix[(old_state, temp_char)]
                break

            return_token += temp_char
            old_state = transition_matrix[(old_state, temp_char)]
            next_char = self.peek()
            if next_char in whitespace or next_char in ("(", ")"):
                break

        return build_token_func(old_state, return_token)

    def scan(self, transition_matrix, build_token_func):
        print "scanning..."
        while not self.eos():
            self.__token_list.append(self.__make_token(
                transition_matrix, build_token_func))
        return self.__token_list

    def pos(self):
        return self.__pos

    def eos(self):
        return self.__pos >= self.__length

    def skip(self, pattern):
        while not self.eos():
            temp_char = self.peek()
            if temp_char in pattern:
                temp_char = self.get()
            else:
                break

    def __skip_whitespace(self):
        self.skip(whitespace)

    def peek(self, length=1):
        return self.__source_string[self.__pos: self.__pos + length]

    def get(self, length=1):
        return_get_string = self.peek(length)
        self.__pos += len(return_get_string)
        return return_get_string


class CuteScanner(object):

    transM = {}

    def __init__(self, source):
        """
        :type source:str
        :param source:
        :return:
        """
        self.source = source
        self._init_TM()

    def _init_TM(self):
        for alpha in letters:
            self.transM[(0, alpha)] = 4
```

```
            self.transM[(4, alpha)] = 4

        for digit in digits:
            self.transM[(0, digit)] = 1
            self.transM[(1, digit)] = 1
            self.transM[(2, digit)] = 1
            self.transM[(4, digit)] = 4

        #Fill Out
        # Complete the remaining transition matrix

    def tokenize(self):
        print "  ===  tokenize  === "

        def build_token(type, lexeme): return Token(type, lexeme)
        print self.source
        cute_scanner = Scanner(self.source)
        return cute_scanner.scan(self.transM, build_token)


test_cute = CuteScanner("Test car + ' - * #T ( ) eq?")
test_tokens = test_cute.tokenize()
print test_tokens
```

유의 사항

- 테스트 함수 고치지 말 것
- 고쳐서 결과가 다른 경우 불이익이 갈 수 있음
- 에러가 있는 input 은 없다고 가정 (그렇다고 과제 의도와 다르게 코딩해도 된다는 것은 절대 아님

결과화면



참고자료 – Dictionary와 for, if문이 포함된 List

Key와 Value로 이루어진 자료형이다. Hash에 가까운 자료형이다. 문자열과 number는 언제나 Key가 될 수 있다. 다른 언어에서는 map과 유사한 자료 구조이다.

Simple Example

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']
```

Updating Example

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};

dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry

print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

파이썬에서는 switch문이 없다. Dictionary는 파이썬에서 switch문 역할을 하기도 한다. 하지만 switch문에서 defualt와 같이 key가 없을 경우에 대해서 처리를 하지 못하기 때문에 주의해서 코딩 해야 한다.

```
def switch_example(i):
    return {
        0: 'Key is 0',
        1: 'Key is 1',
        2: 'Key is 2',
    }[i]

print switch_example(0)
```

파이썬에서는 list를 반복문과 조건문을 포함하여 생성할 수 있다.

```
squares = []
for x in range(10):
    squares.append(x)

print squares
```

위와 같이 생성된 list를 다음과 같은 표현으로 생성할 수 있다.

```
squares = [x for x in range(10)]
print squares
```

다음은 반복문과 조건문을 혼합하여 쓴 예이다.

```
combs = [(x, y) for x in [1, 2, 3] for y in [3, 1, 4] if x != y]
print combs
```

위의 표현은 아래와 같다.

```
combs = []
for x in [1, 2, 3]:
    for y in [3, 1, 4]:
        if x!=y:
            combs.append((x, y))
print combs
```

이와 같은 표현은 list뿐만 아니라 dictionary에서도 사용 가능하다. 위의 조건문은 변수 assignment시에도 사용이 가능하다.

```
a = 3
b = 2 if a is 3 else 4
print b
```

위의 표현은 아래와 같다.

```
if a is 3:
    b = 2
else : b = 4
```

출처: http://www.tutorialspoint.com/python/python_dictionary.htm

https://docs.python.org/2/tutorial/datastructures.html?highlight=dictionary

수정: 2017-03-18